RUBY CLASSES AND OBJECTS

Ruby is a perfect Object Oriented Programming Language. The features of the object-oriented programming language include:

- Data Encapsulation:
- Data Abstraction:
- Polymorphism:
- Inheritance:

These features have been discussed in Object Oriented Ruby.

An object-oriented program involves classes and objects. A class is the blueprint from which individual objects are created. In object-oriented terms, we say that your *bicycle* is an instance of the *class of objects* known as bicycles.

Take the example of any vehicle. It comprises wheels, horsepower, and fuel or gas tank capacity. These characteristics form the data members of the class Vehicle. You can differentiate one vehicle from the other with the help of these characteristics.

A vehicle can also have certain functions, such as halting, driving, and speeding. Even these functions form the data members of the class Vehicle. You can, therefore, define a class as a combination of characteristics and functions.

A class Vehicle can be defined as:

```
Class Vehicle
{
   Number no_of_wheels
   Number horsepower
   Characters type_of_tank
   Number Capacity
   Function speeding
   {
   }
   Function driving
   {
   }
   Function halting
   {
   }
}
```

By assigning different values to these data members, you can form several instances of the class Vehicle. For example, an airplane has three wheels, horsepower of 1,000, fuel as the type of tank, and a capacity of 100 liters. In the same way, a car has four wheels, horsepower of 200, gas as the type of tank, and a capacity of 25 litres.

Defining a Class in Ruby:

To implement object-oriented programming by using Ruby, you need to first learn how to create objects and classes in Ruby.

A class in Ruby always starts with the keyword *class* followed by the name of the class. The name should always be in initial capitals. The class *Customer* can be displayed as:

```
class Customer
```

```
end
```

You terminate a class by using the keyword *end*. All the data members in the *class* are between the class definition and the *end* keyword.

Variables in a Ruby Class:

Ruby provides four types of variables:

- **Local Variables:** Local variables are the variables that are defined in a method. Local variables are not available outside the method. You will see more detail about method in subsequent chapter. Local variables begin with a lowercase letter or _.
- **Instance Variables:** Instance variables are available across methods for any particular instance or object. That means that instance variables change from object to object. Instance variables are preceded by the at sign (@) followed by the variable name.
- **Class Variables:** Class variables are available across different objects. A class variable belongs to the class and is a characteristic of a class. They are preceded by the sign @@ and are followed by the variable name.
- **Global Variables:** Class variables are not available across classes. If you want to have a single variable, which is available across classes, you need to define a global variable. The global variables are always preceded by the dollar sign ($).

Example:

Using the class variable @@no_of_customers, you can determine the number of objects that are being created. This enables in deriving the number of customers.

```
class Customer
   @@no_of_customers=0
end
```

Creating Objects in Ruby using *new* Method:

Objects are instances of the class. You will now learn how to create objects of a class in Ruby. You can create objects in Ruby by using the method *new* of the class.

The method *new* is a unique type of method, which is predefined in the Ruby library. The new method belongs to the *class* methods.

Here is the example to create two objects cust1 and cust2 of the class Customer:

```
cust1 = Customer. new
cust2 = Customer. new
```

Here, cust1 and cust2 are the names of two objects. You write the object name followed by the equal to sign (=) after which the class name will follow. Then, the dot operator and the keyword *new* will follow.

Custom Method to create Ruby Objects :

You can pass parameters to method *new* and those parameters can be used to initialize class variables.

When you plan to declare the *new* method with parameters, you need to declare the method *initialize* at the time of the class creation.

The *initialize* method is a special type of method, which will be executed when the *new* method of the class is called

with parameters.

Here is the example to create initialize method:

```
class Customer
   @@no_of_customers=0
   def initialize(id, name, addr)
      @cust_id=id
      @cust_name=name
      @cust_addr=addr
   end
end
```

In this example, you declare the *initialize* method with **id, name**, and **addr** as local variables. Here *def* and *end* are used to define a Ruby method *initialize*. You will learn more about methods in subsequent chapters.

In the *initialize* method, you pass on the values of these local variables to the instance variables @cust_id, @cust_name, and @cust_addr. Here local variables hold the values that are passed along with the new method.

Now you can create objects as follows:

```
cust1=Customer.new("1", "John", "Wisdom Apartments, Ludhiya")
cust2=Customer.new("2", "Poul", "New Empire road, Khandala")
```

Member Functions in Ruby Class:

In Ruby, functions are called methods. Each method in a *class* starts with the keyword *def* followed by the method name.

The method name always preferred in **lowercase letters**. You end a method in Ruby by using the keyword *end*.

Here is the example to define a Ruby method:

```
class Sample
   def function
      statement 1
      statement 2
   end
end
```

Here *statement 1* and *statement 2* are part of the body of the method *function* inside the class Sample. These statments could be any valid Ruby statement. For example we can put a method *puts* to print *Hello Ruby* as follows:

```
class Sample
   def hello
      puts "Hello Ruby!"
   end
end
```

Now in the following example create one object of Sample class and call *hello* method and see the result:

```
#!/usr/bin/ruby

class Sample
   def hello
      puts "Hello Ruby!"
   end
end

# Now using above class to create objects
object = Sample. new
object.hello
```

This will produce following result:

```
Hello Ruby!
```

Simple Case Study:

Here is a case study if you want to do more practice with class and objects:

Ruby Class Case Study